Introduction to TCP/IP networking

TCP/IP protocol family

- IP : Internet Protocol
  - UDP : User Datagram Protocol
    - RTP, traceroute
  - TCP : Transmission Control Protocol
    - HTTP, FTP, ssh

What is an internet?

- A set of *inter*connected *net*works
- The **I**nternet is the most famous example

- Networks can be completely different
  - Ethernet, ATM, modem, …
  - (TCP/)IP is what links them

What is an internet? (cont)

- *Routers* (nodes) are devices on multiple networks that pass traffic between them
- Individual networks pass traffic from one router or endpoint to another
- TCP/IP hides the details as much as possible

ISO/OSI Network Model (Don't need to know this)

- Seven network "layers"
  - Layer 1 : Physical – cables
  - Layer 2 : Data Link – ethernet
  - Layer 3 : Network – IP
  - Layer 4 : Transport – TCP/UDP
  - Layer 5 : Session
  - Layer 6 : Presentation
  - Layer 7 : Application

You don't need to know the layers just the idea that it is layered

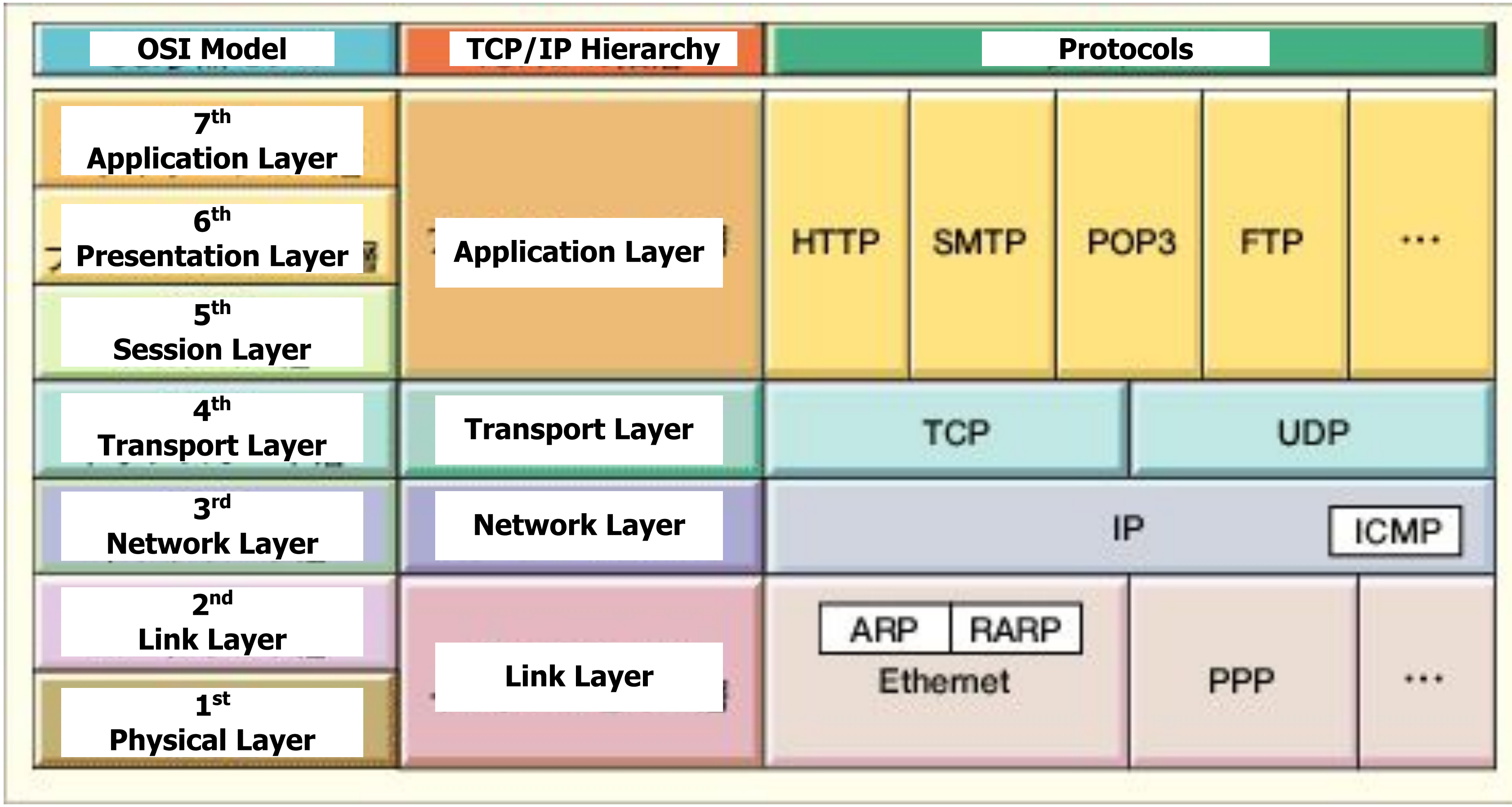
TCP/IP Network Model

- Different view – 4 layers
  - Layer 1 : Link (we did not look at details)
  - Layer 2 : Network
  - Layer 3 : Transport
  - Layer 4 : Application

Ethernet

- Data Link Layer protocol
- Ethernet (IEEE 802.3) is widely used.
- Supported by a variety of physical layer implementations.
- Multi-access (shared medium).

OSI: Open Systems Interconnect



Link Layer : includes device driver and network interface card
Network Layer : handles the movement of packets, i.e. Routing
Transport Layer : provides a reliable flow of data between two hosts
Application Layer : handles the details of the particular application

CSMA/CD

- *Carrier Sense Multiple Access with Collision Detection*

- *Carrier Sense* : can tell when another host is transmitting

- *Multiple Access* : many hosts on 1 wire

- *Collision Detection* : can tell when another host transmits at the same time.

An Ethernet Frame

Preamble	Destination Address	Source Address	Len	DATA	CRC
8 bytes	6	6	2	0-1500	4

- The preamble is a sequence of alternating 1s and 0s used for synchronization.
- CRC is Cyclic Redundancy Check

Ethernet Addressing

- Every Ethernet interface has a unique 48 bit address (a.k.a. *hardware address*).
  - Example: **`C0:B3:44:17:21:17`**
  - The broadcast address is all 1's.
  - Addresses are assigned to vendors by a central authority.

- Each interface looks at every *frame* and inspects the destination address. If the address does not match the hardware address of the interface (or the broadcast address), the frame is discarded.

Internet Protocol

- IP is the network layer
  - packet delivery service (host-to-host).
  - translation between different data-link protocols

- IP provides connectionless, unreliable delivery of *IP datagrams*.
  - *Connectionless*: each datagram is independent of all others.
  - *Unreliable*: there is no guarantee that datagrams are delivered correctly or even delivered at all.

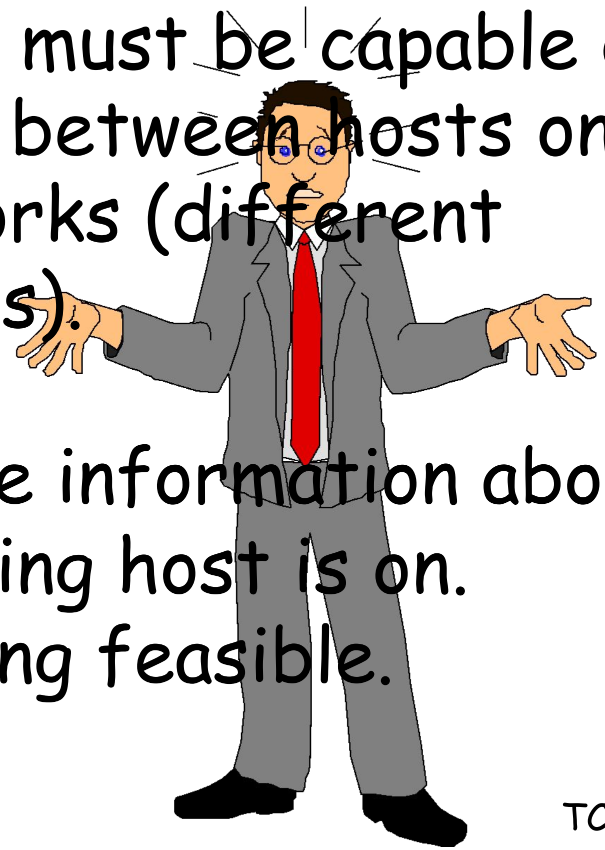
IP

- Responsible for end to end transmission
- Sends data in individual packets
- Maximum size of packet is determined by the networks
  - Fragmented if too large
- Unreliable
  - Packets might be lost, corrupted, duplicated, delivered out of order

IP Addresses

- IP addresses are not the same as the underlying data-link (MAC) addresses.

- IP is a network layer - it must be capable of providing communication between hosts on different kinds of networks (different data-link implementations).



- The address must include information about what *network* the receiving host is on. This is what makes routing feasible.

IP Addresses

- IP addresses are *logical* addresses (not physical)
- 32 bits. ← IPv4 *(version 4)*
- Includes a network ID and a host ID.
- Every host must have a unique IP address.
- IP addresses are assigned by a central authority (*American Registry for Internet Numbers* for North America).

IP addresses

- 4 bytes
  - e.g. 163.1.125.98
  - Each device normally gets one (or more)
  - In theory there are about 4 billion available
- But…

The four formats of IP Addresses

Class

A | 0 | NetID | HostID |

128 possible network IDs, over 4 million host IDs per network ID

B | 10 | NetID | HostID |

16K possible network IDs, 64K host IDs per network ID

C | 110 | NetID | HostID |

Over 2 million possible network IDs, 256 host IDs per network ID

D | 1110 | Multicast Address |

8 bits 8 bits 8 bits 8 bits

Network and Host IDs

- A Network ID is assigned to an organization by a global authority.
- Host IDs are assigned locally by a system administrator.
- Both the Network ID and the Host ID are used for routing.

IP Addresses

- IP Addresses are usually shown in *dotted decimal* notation:

  1.2.3.4

  00000001 00000010 00000011 00000100

- cse.unr.edu is 134.197.40.3

  10000110 11000101 00101000 00000010

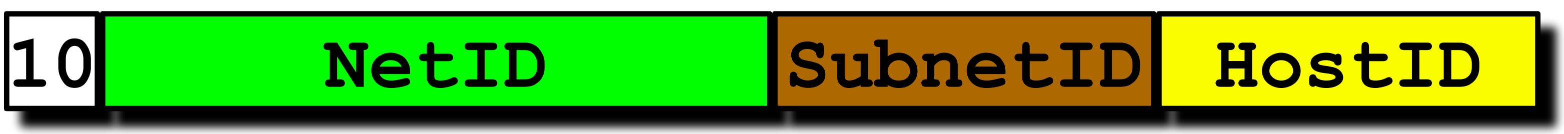
CSE has a class B network

Host and Network Addresses

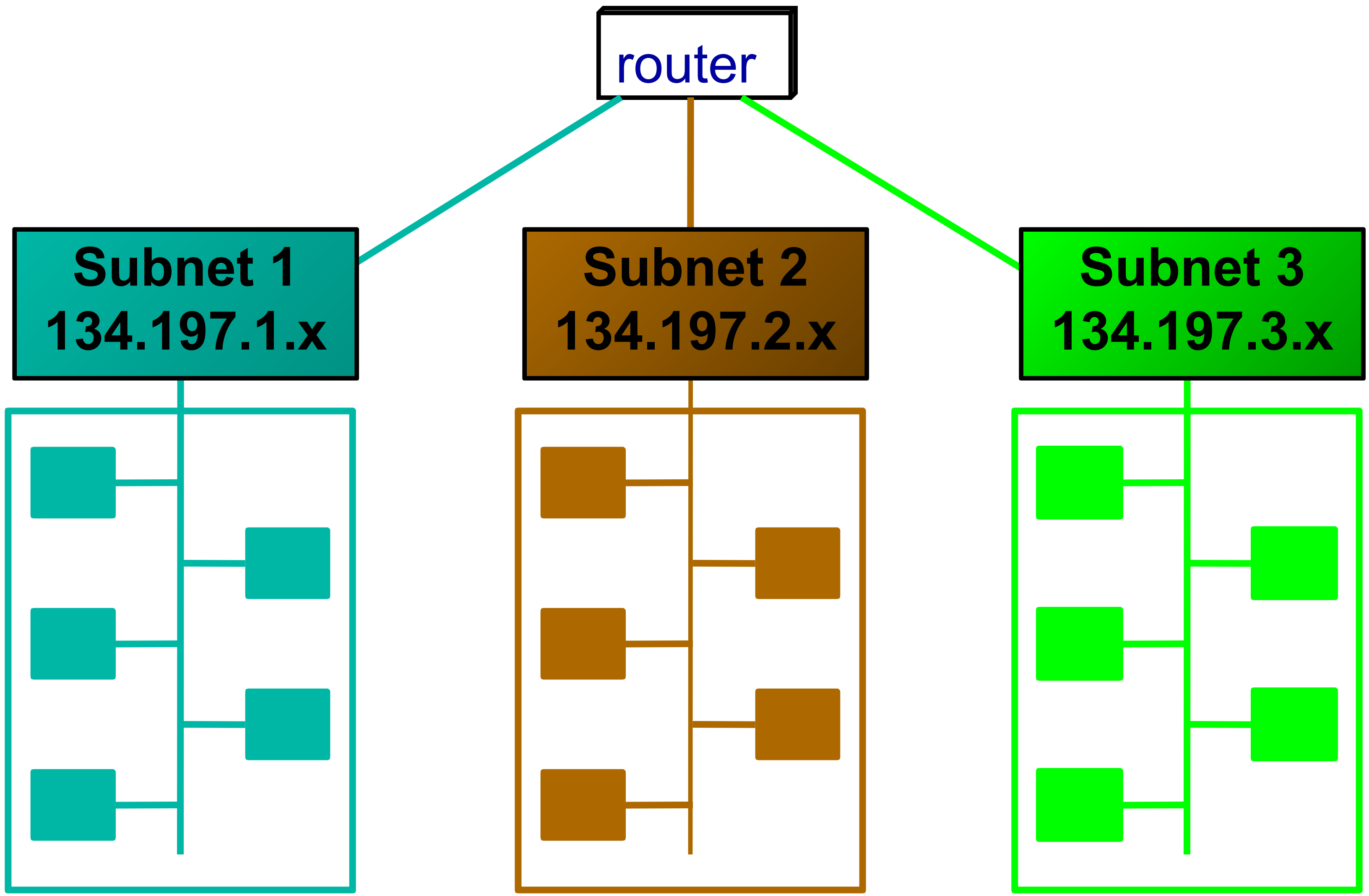
- A single network interface is assigned a single IP address called the *host* address.
- A host may have multiple interfaces, and therefore multiple *host* addresses.
- Hosts that share a network all have the same IP *network* address (the network ID).
- An IP address that has a host ID of all 0s is called a *network address* and refers to an entire network.

Subnet Addresses

- An organization can subdivide it's host address space into groups called subnets.
- The subnet ID is generally used to group hosts based on the physical network topology.



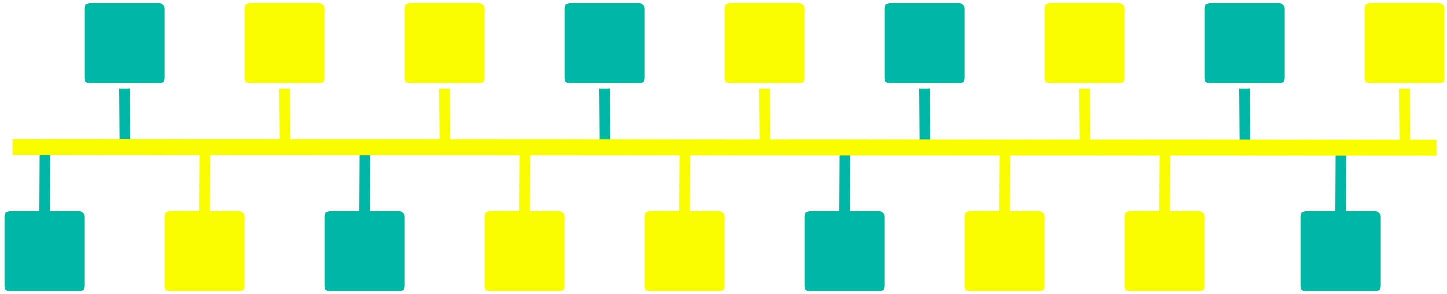
Subnetting



router
Subnet 1
134.197.1.x
Subnet 2
134.197.2.x
Subnet 3
134.197.3.x

Subnetting

- Subnets can simplify routing.
- IP subnet broadcasts have a hostID of all 1s.
- It is possible to have a single wire network with multiple subnets.



Routing

- How does a device know where to send a packet?
  - All devices need to know what IP addresses are on directly attached networks
  - If the destination is on a local network, send it directly there

Routing (cont)

- If the destination address isn't local
  - Most non-router devices just send everything to a single local router
  - Routers need to know which network corresponds to each possible IP address

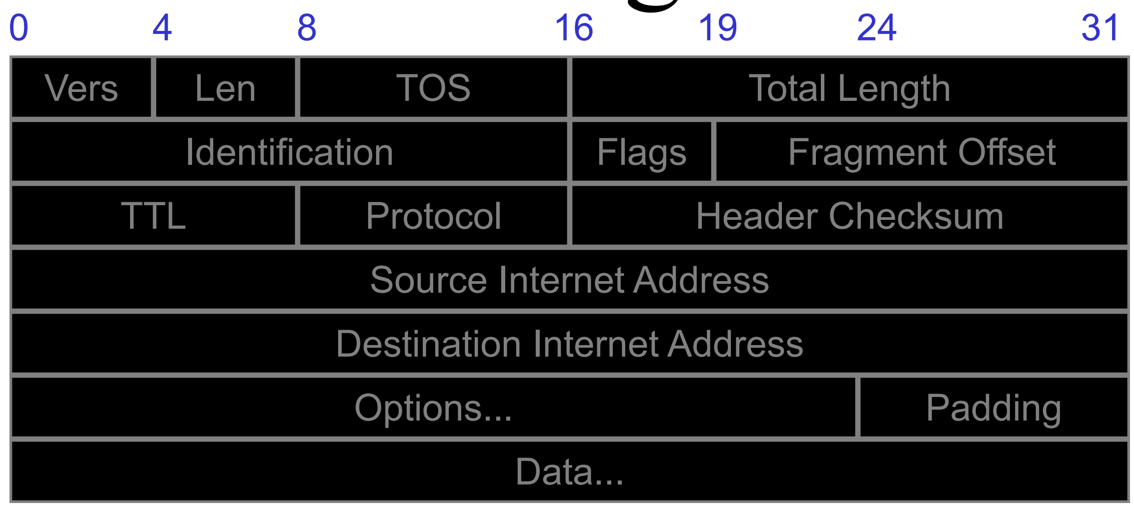
Allocation of addresses

- Controlled centrally by ICANN
  - Fairly strict rules on further delegation to avoid wastage
    - Have to demonstrate actual need for them
- Organizations that got in early have bigger allocations than they really need

IP packets

- Source and destination addresses
- Protocol number
  - 1 = ICMP, 6 = TCP, 17 = UDP
- Various options
  - e.g. to control fragmentation
- Time to live (TTL)
  - Prevent routing loops

IP Datagram

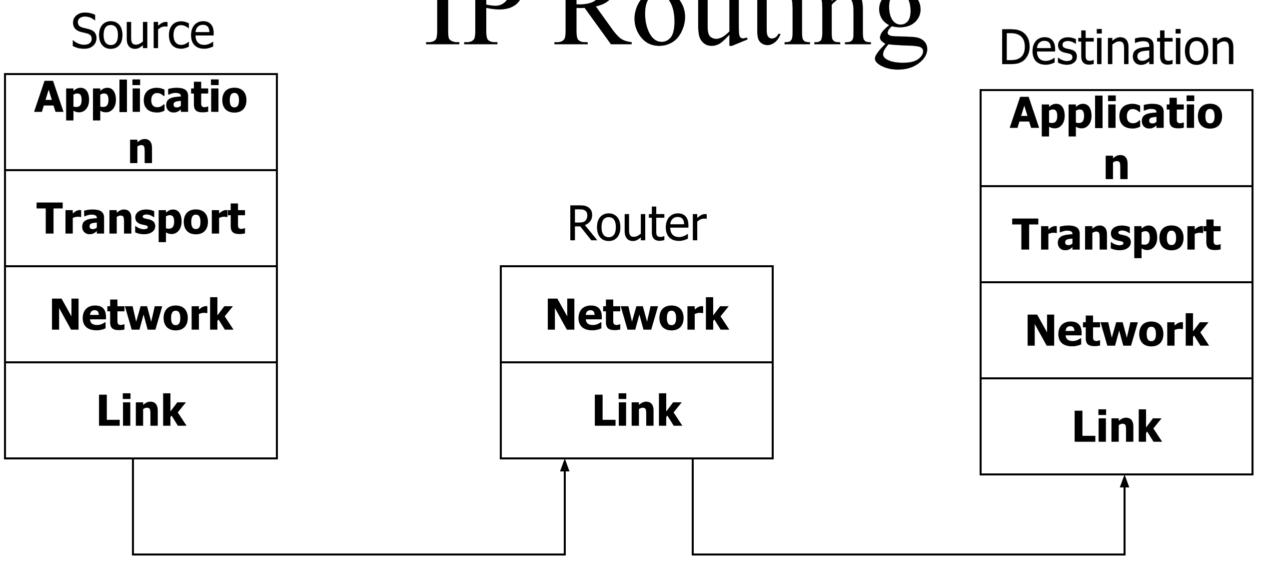


Field	**Purpose**
Vers	IP version number
Len	Length of IP header (4 octet units)
TOS	Type of Service
T. Length	Length of entire datagram (octets)
Ident.	IP datagram ID (for frag/reassembly)
Flags	Don't/More fragments
Frag Off	Fragment Offset

Field	**Purpose**
TTL	Time To Live - Max # of hops
Protocol	Higher level protocol (1=ICMP, 6=TCP, 17=UDP)
Checksum	Checksum for the IP header
Source IA	Originator's Internet Address
Dest. IA	Final Destination Internet Address
Options	Source route, time stamp, etc.
Data...	Higher level protocol data

We only looked at the IP addresses, TTL and protocol #

IP Routing



- Routing Table

  Destination IP address

  IP address of a next-hop router

  Flags

  Network interface specification

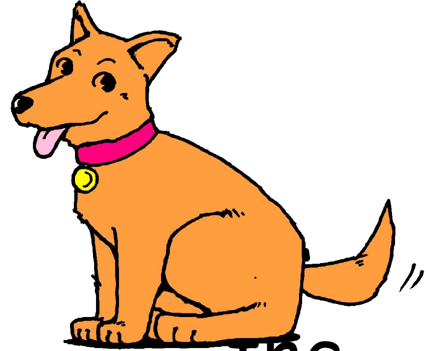
Mapping IP Addresses to Hardware Addresses

- IP Addresses are not recognized by hardware.
- If we know the IP address of a host, how do we find out the hardware address ?
- The process of finding the hardware address of a host given the IP address is called

Address Resolution

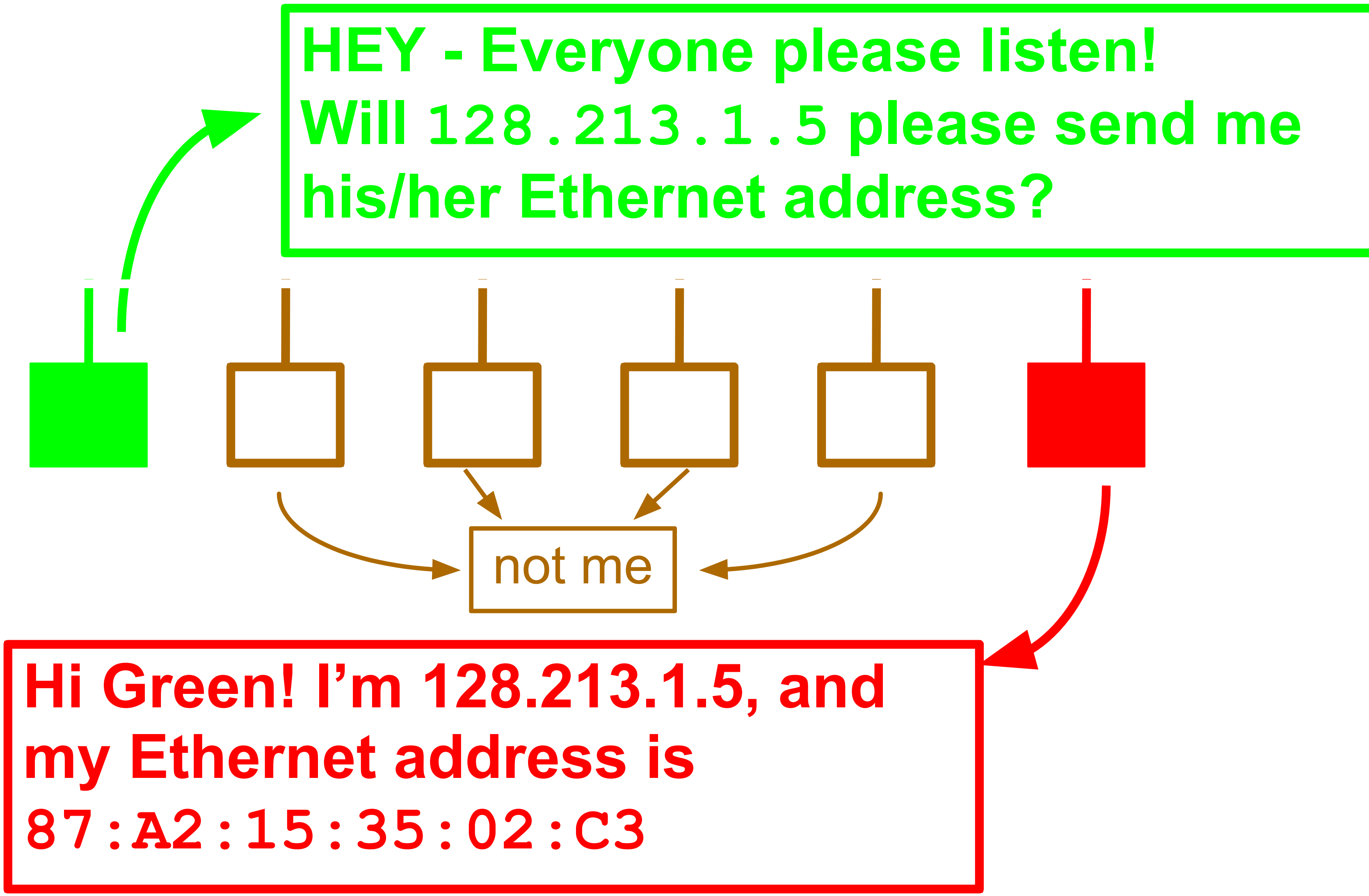
ARP

Arp Arp!



- The *Address Resolution Protocol* used by a sending host when it knows the IP address of the destination but needs the Ethernet (or whatever) address.
- ARP is a broadcast protocol - every host on the network receives the request.
- Each host checks the request against it's IP address - the right one responds.
- hosts *remember* the hardware addresses of each other.

ARP conversation



IP Datagram

<table>
<tr><td colspan="2">1 byte</td><td colspan="2">1 byte</td><td colspan="2">1 byte</td><td colspan="2">1 byte</td></tr>
<tr><td>VERS</td><td>HL</td><td colspan="2">Service</td><td colspan="4">Fragment Length</td></tr>
<tr><td colspan="4">Datagram ID</td><td>FLAG</td><td colspan="3">Fragment Offset</td></tr>
<tr><td colspan="2">TTL</td><td colspan="2">Protocol</td><td colspan="4">Header Checksum</td></tr>
<tr><td colspan="8">Source Address</td></tr>
<tr><td colspan="8">Destination Address</td></tr>
<tr><td colspan="8">Options (if any)</td></tr>
<tr><td colspan="8">Data</td></tr>
</table>

IP Datagram Fragmentation

- Packets are fragmented due to link's Maximum Transmission Unit (MTU)
- Each fragment (packet) has the same structure as the IP datagram.
- IP specifies that datagram reassembly is done only at the destination (not on a hop-by-hop basis).
- If any of the fragments are lost - the entire datagram is discarded (and an ICMP message is sent to the sender).

IP Flow Control & Error Detection

- If packets arrive too fast - the receiver discards excessive packets and sends an ICMP message to the sender (SOURCE QUENCH).
- If an error is found (header checksum problem) the packet is discarded and an ICMP message is sent to the sender.

ICMP
Internet Control Message Protocol

- ICMP is a protocol used for exchanging control messages.
- ICMP uses IP to deliver messages.
- ICMP messages are usually generated and processed by the IP software, not the user process.

ICMP Message Types

- Echo Request
- Echo Response
- Destination Unreachable
- Redirect
- Time Exceeded
- Redirect (route change)
- there are more ...

IPv6

- 128 bit addresses
  - Make it feasible to be very wasteful with address allocations
- Lots of other new features
  - Built-in autoconfiguration, security options, …
- Not really in production use yet

Transport Layer & TCP/IP

Q: We know that IP is the network layer
- so TCP must be the transport layer, right ?

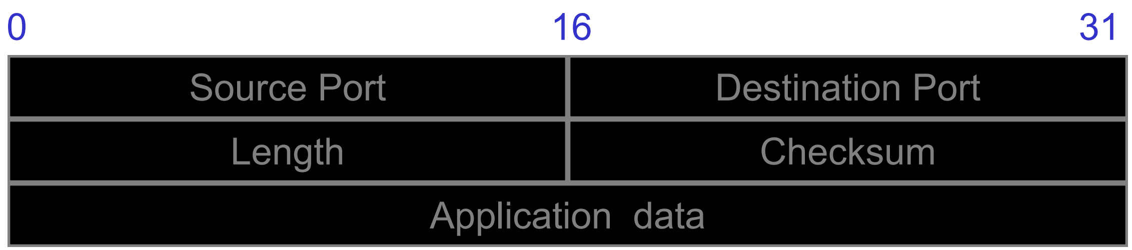
A: No… well, almost.

TCP is only part of the TCP/IP transport layer
- the other part is UDP (User Datagram Protocol).

UDP

- Thin layer on top of IP
- Adds packet length + checksum
  - Guard against corrupted packets
- Also source and destination *ports*
  - Ports are used to associate a packet with a specific application at each end
- Still unreliable:
  - Duplication, loss, out-of-orderness possible

UDP datagram



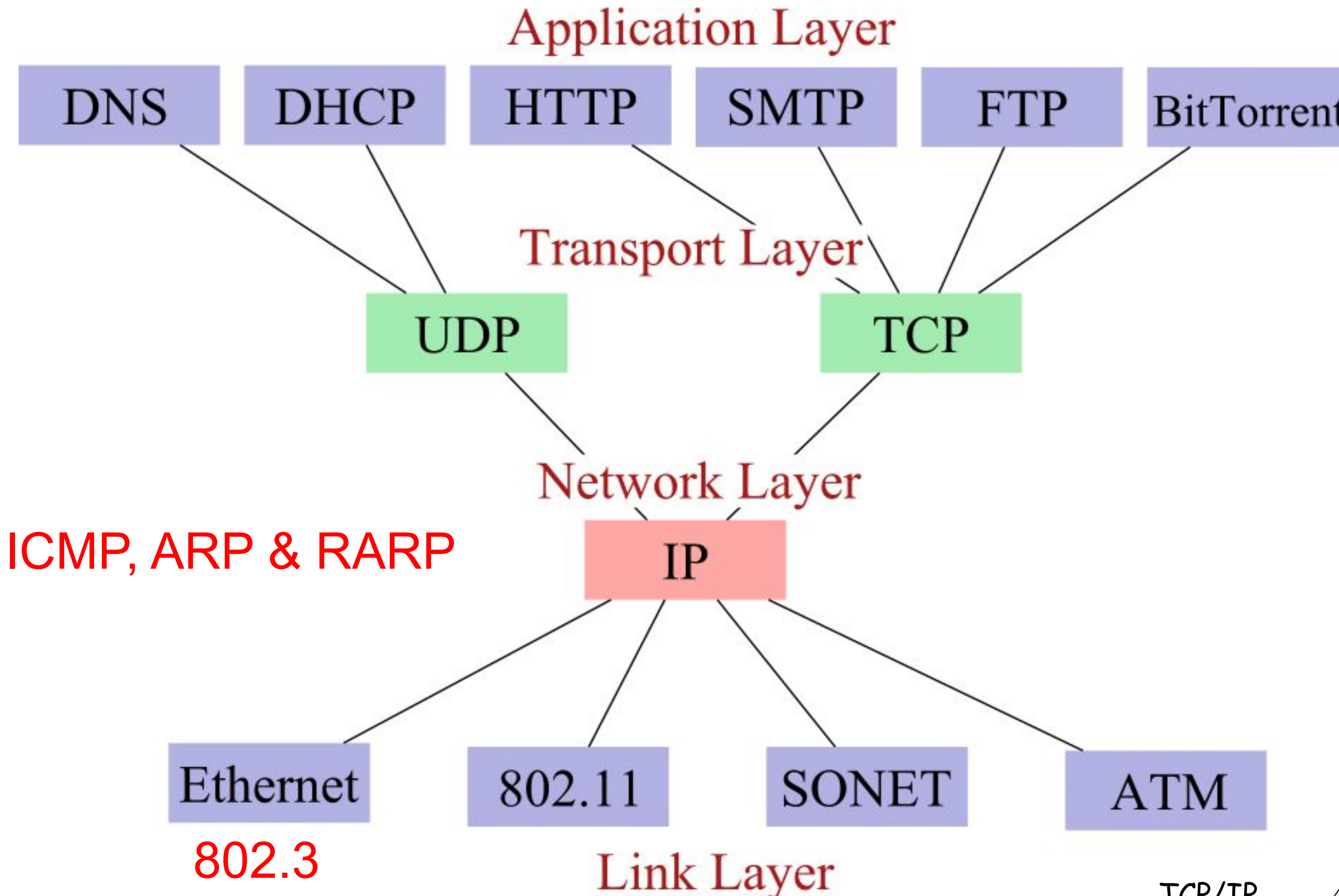
Field	Purpose
Source Port	16-bit port number identifying originating application
Destination Port	16-bit port number identifying destination application
Length	Length of UDP datagram (UDP header + data)
Checksum	Checksum of IP pseudo header, UDP header, and data

Typical applications of UDP

- Where packet loss etc is better handled by the application than the network stack
- Where the overhead of setting up a connection isn't wanted

- VOIP
- NFS – Network File System
- Most games

The Internet Hourglass

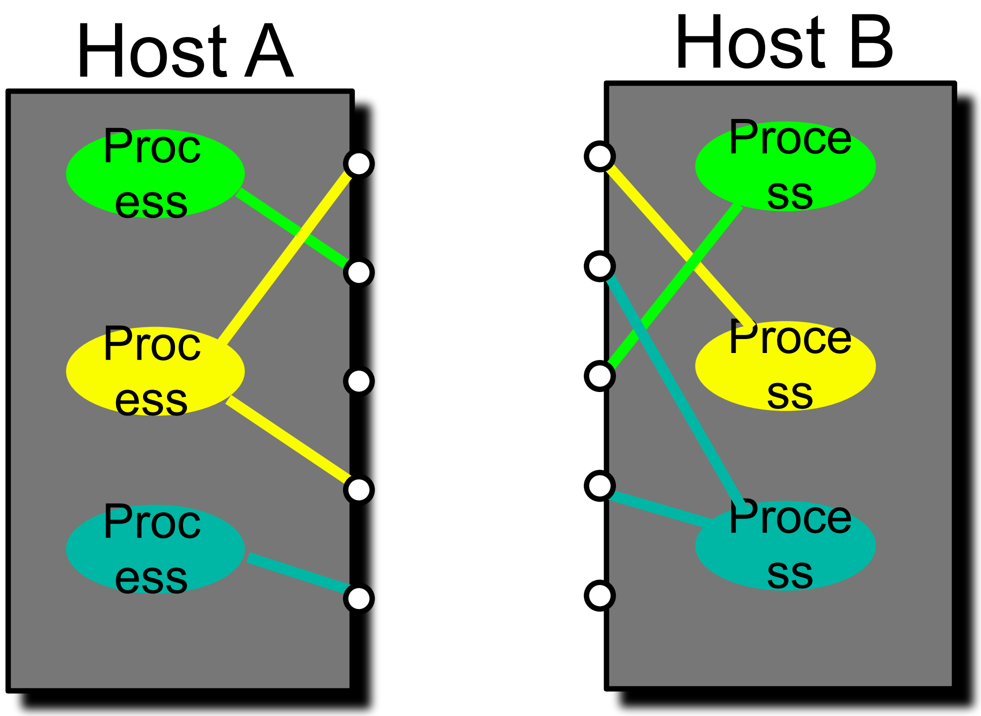


UDP User Datagram Protocol

- UDP is a transport protocol
  - communication between processes
- UDP uses IP to deliver datagrams to the right host.
- UDP uses *ports* to provide communication services to individual processes.

Ports

- TCP/IP uses an abstract destination point called a protocol port.
- Ports are identified by a positive integer.
- Operating systems provide some mechanism that processes use to specify a port.



UDP

- Datagram Delivery
- Connectionless
- Unreliable
- Minimal

UDP Datagram Format

Source Port	Destination Port
Length	Checksum
Data	

TCP
Transmission Control Protocol

- TCP is an alternative transport layer protocol supported by TCP/IP.

- TCP provides:
  - Connection-oriented
  - Reliable
  - Full-duplex
  - Byte-Stream

Connection-Oriented

- *Connection oriented* means that a virtual connection is established before any user data is transferred.
- If the connection cannot be established, the user program is notified (finds out).
- If the connection is ever interrupted, the user program(s) is finds out there is a problem.

Reliable

- *Reliable* means that every transmission of data is acknowledged by the receiver.
- Reliable does not mean that things don't go wrong, it means that we find out when things go wrong.
- If the sender does not receive acknowledgement within a specified amount of time, the sender retransmits the data.

Byte Stream

- *Stream* means that the connection is treated as a stream of bytes.
- The user application does not need to package data in individual datagrams (as with UDP).

Buffering

- TCP is responsible for buffering data and determining when it is time to send a datagram.
- It is possible for an application to tell TCP to send the data it has buffered without waiting for a buffer to fill up.

Full Duplex

- TCP provides transfer in both directions (over a single virtual connection).
- To the application program these appear as 2 unrelated data streams, although TCP can piggyback control and data communication by providing control information (such as an ACK) along with user data.

TCP

- Reliable, *full-duplex*, *connection-oriented*, *stream* delivery
  - Interface presented to the application doesn’t require data in individual packets
  - Data is guaranteed to arrive, and in the correct order without duplications
    - Or the connection will be dropped
  - Imposes significant overheads

Applications of TCP

- Most things!
  - HTTP, FTP, …

- Saves the application a lot of work, so used unless there’s a good reason not to

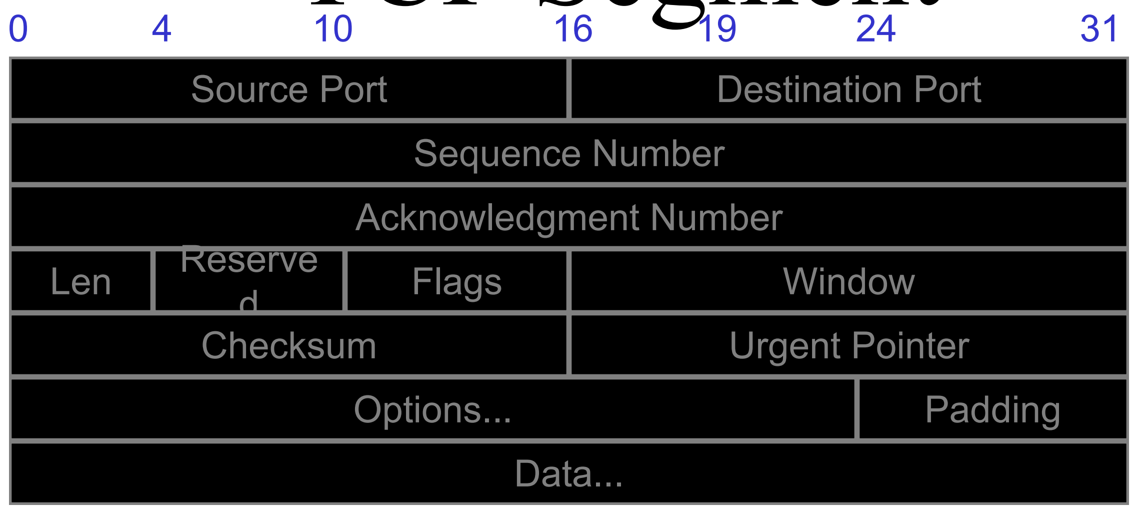
TCP implementation

- Connections are established using a *three-way handshake*
- Data is divided up into packets by the operating system
- Packets are numbered, and received packets are acknowledged
- Connections are explicitly closed
  – (or may abnormally terminate)

TCP Packets

- Source + destination ports
- Sequence number (used to order packets)
- Acknowledgement number (used to verify packets are received)

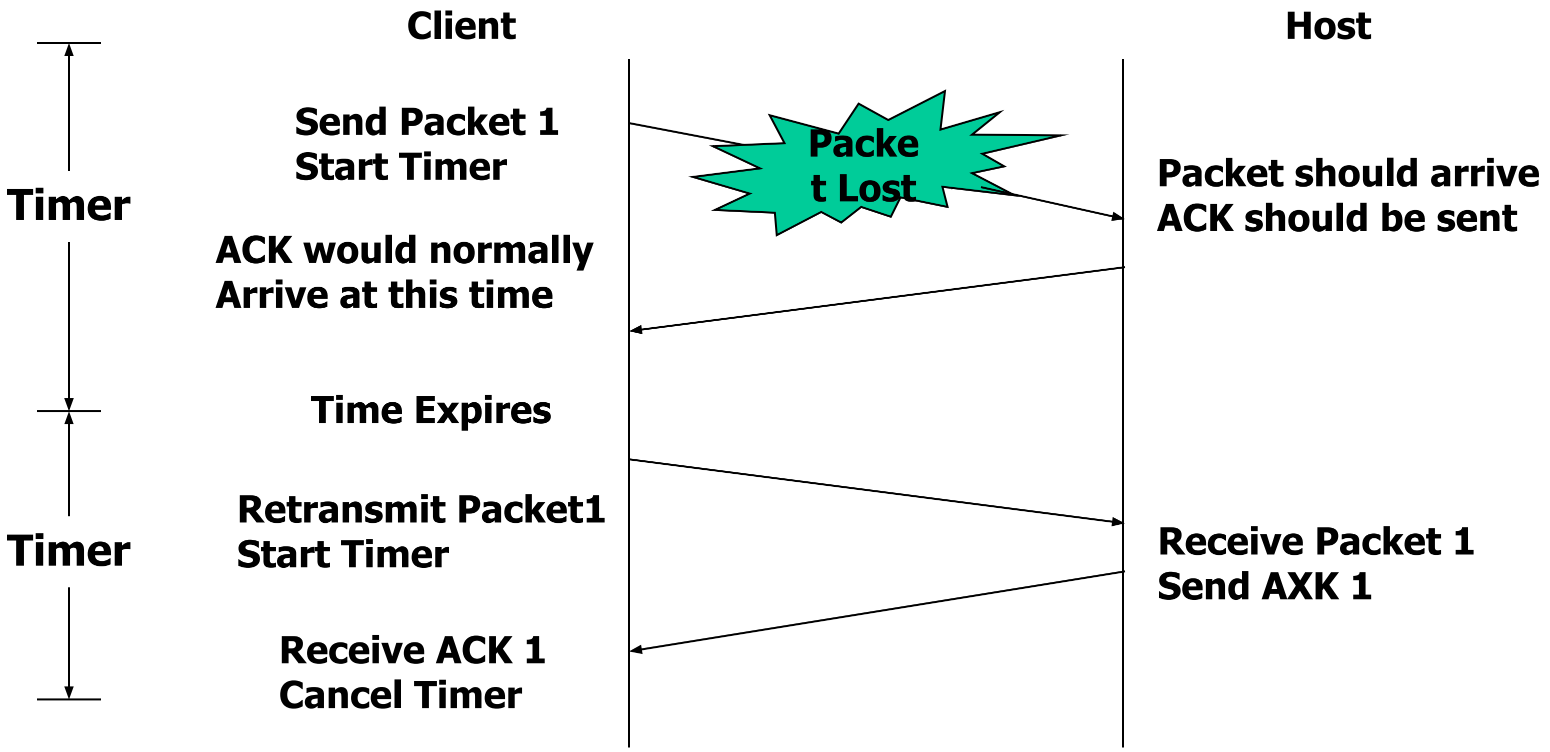
TCP Segment



Field	**Purpose**
Source Port	Identifies originating application
Destination Port	Identifies destination application
Sequence Number	Sequence number of first octet in the segment
Acknowledgment #	Sequence number of the next expected octet (if ACK flag set)
Len	Length of TCP header in 4 octet units
Flags	TCP flags: SYN, FIN, RST, PSH, ACK, URG
Window	Number of octets from ACK that sender will accept
Checksum	Checksum of IP pseudo-header + TCP header + data
Urgent Pointer	Pointer to end of “urgent data”
Options	Special TCP options such as MSS and Window Scale

You just need to know port numbers, seq and ack are added

TCP : Data transfer



TCP Ports

- Interprocess communication via TCP is achieved with the use of ports (just like UDP).
- UDP ports have no relation to TCP ports (different name spaces).

TCP Segments

- The chunk of data that TCP asks IP to deliver is called a *TCP segment.*

- Each segment contains:
  - data bytes from the byte stream
  - control information that identifies the data bytes

TCP Segment Format

<table>
<tr><td colspan="2">1 byte</td><td colspan="2">1 byte</td><td colspan="2">1 byte</td><td colspan="2">1 byte</td></tr>
<tr><td colspan="4">Source Port</td><td colspan="4">Destination Port</td></tr>
<tr><td colspan="8">Sequence Number</td></tr>
<tr><td colspan="8">Request Number</td></tr>
<tr><td>offset</td><td>Reser.</td><td colspan="2">Control</td><td colspan="4">Window</td></tr>
<tr><td colspan="4">Checksum</td><td colspan="4">Urgent Pointer</td></tr>
<tr><td colspan="8">Options (if any)</td></tr>
<tr><td colspan="8">Data</td></tr>
</table>

Addressing in TCP/IP

- Each TCP/IP address includes:
  - Internet Address
  - Protocol (UDP or TCP)
  - Port Number

NOTE: TCP/IP is a *protocol suite* that includes IP, TCP and UDP

TCP vs. UDP

Q: Which protocol is better ?

A: It depends on the application.

TCP provides a connection-oriented, reliable, byte stream service (lots of overhead).

UDP offers minimal datagram delivery service (as little overhead as possible).

TCP Lingo

- When a client requests a connection, it sends a "SYN" segment (a special TCP segment) to the server port.
- SYN stands for *synchronize*. The SYN message includes the client's ISN.
- ISN is Initial Sequence Number.

More...

- Every TCP segment includes a *Sequence Number* that refers to the first byte of *data* included in the segment.

- Every TCP segment includes a *Request Number* (*Acknowledgement Number*) that indicates the byte number of the next data that is expected to be received.
  - *All* bytes up through this number have already been received.

And more...

- There are a bunch of control flags:
  - URG: urgent data included.
  - ACK: this segment is (among other things) an acknowledgement.
  - RST: error - abort the session.
  - SYN: synchronize Sequence Numbers (setup)
  - FIN: polite connection termination.

And more...

- MSS: Maximum segment size (A TCP option)

- Window: Every ACK includes a Window field that tells the sender how many bytes it can send before the receiver will have to toss it away (due to fixed buffer size).

TCP Connection Creation

- Programming details later - for now we are concerned with the actual communication.

- A *server* accepts a connection.
  - Must be looking for new connections!

- A *client* requests a connection.
  - Must *know* where the server is!

Client Starts

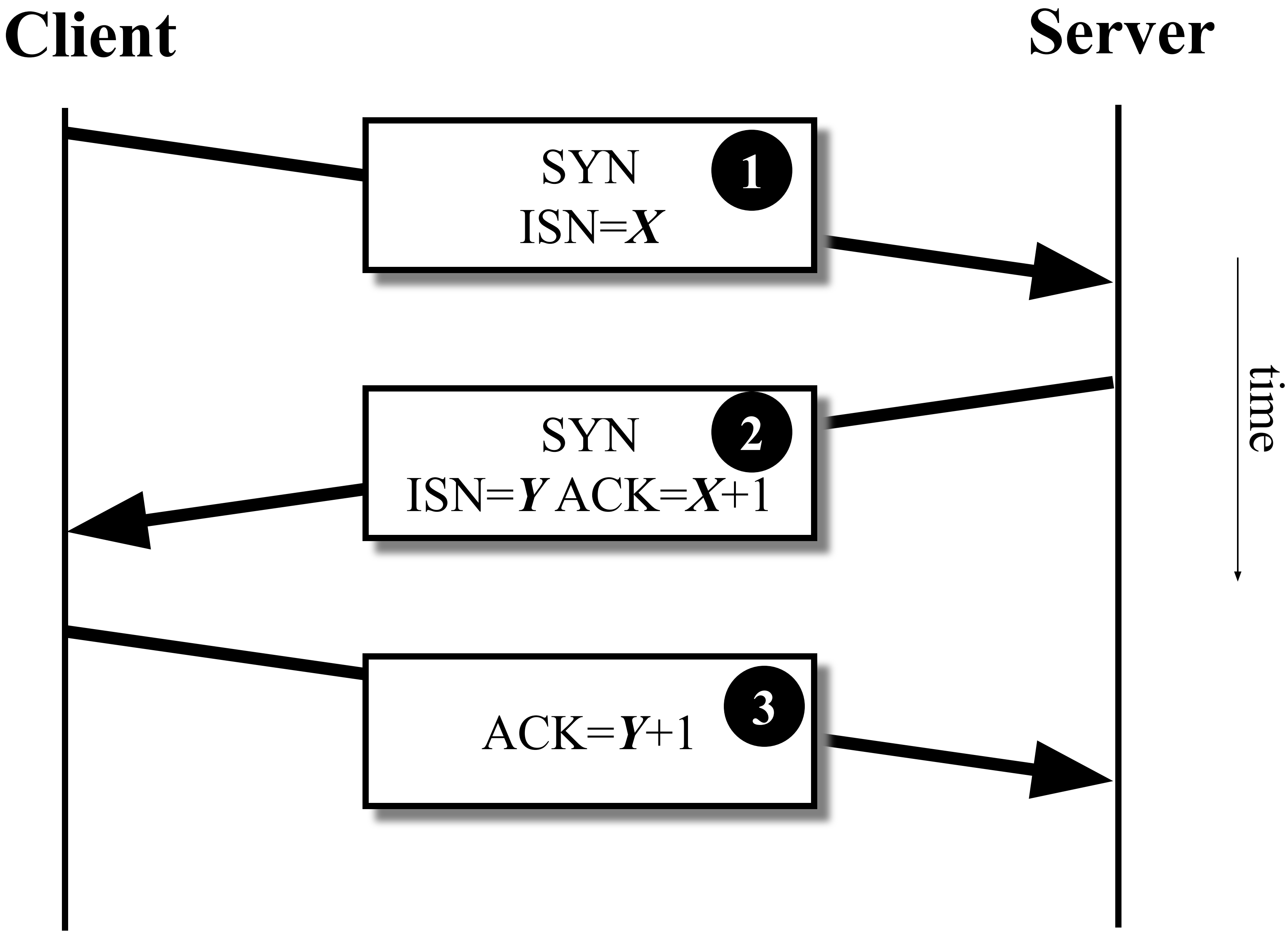
- A client starts by sending a SYN segment with the following information:
  - Client's ISN (generated pseudo-randomly)
  - Maximum Receive Window for client.
  - Optionally (but usually) MSS (largest datagram accepted).
  - No payload! (Only TCP headers)

Sever Response

- When a waiting server sees a new connection request, the server sends back a SYN segment with:
  - Server's ISN (generated pseudo-randomly)
  - Request Number is Client ISN+1
  - Maximum Receive Window for server.
  - Optionally (but usually) MSS
  - No payload! (Only TCP headers)

Finally

- When the Server's SYN is received, the client sends back an ACK with:
  - Request Number is Server's ISN+1


Client
Server
SYN
ISN=X
1
SYN
ISN=Y ACK=X+1
2
ACK=Y+1
3
time

TCP Data and ACK

- Once the connection is established, data can be sent.
- Each data segment includes a sequence number identifying the first byte in the segment.
- Each segment (data or empty) includes a request number indicating what data has been received.

TCP Buffers

- The TCP layer doesn't know when the application will ask for any received data.
  - TCP buffers incoming data so it's ready when we ask for it.
- Both the client and server allocate buffers to hold incoming and outgoing data
  - The TCP layer does this.
- Both the client and server announce with every ACK how much buffer space remains (the Window field in a TCP segment).

Send Buffers

- The application gives the TCP layer some data to send.

- The data is put in a send buffer, where it stays until the data is ACK'd.
  - it has to stay, as it might need to be sent again!

- The TCP layer won't accept data from the application unless (or until) there is buffer space.

ACKs

- A receiver doesn't have to ACK every segment (it can ACK many segments with a single ACK segment).
- Each ACK can also contain outgoing data (piggybacking).
- If a sender doesn't get an ACK after some time limit (MSL) it resends the data.

TCP Segment Order

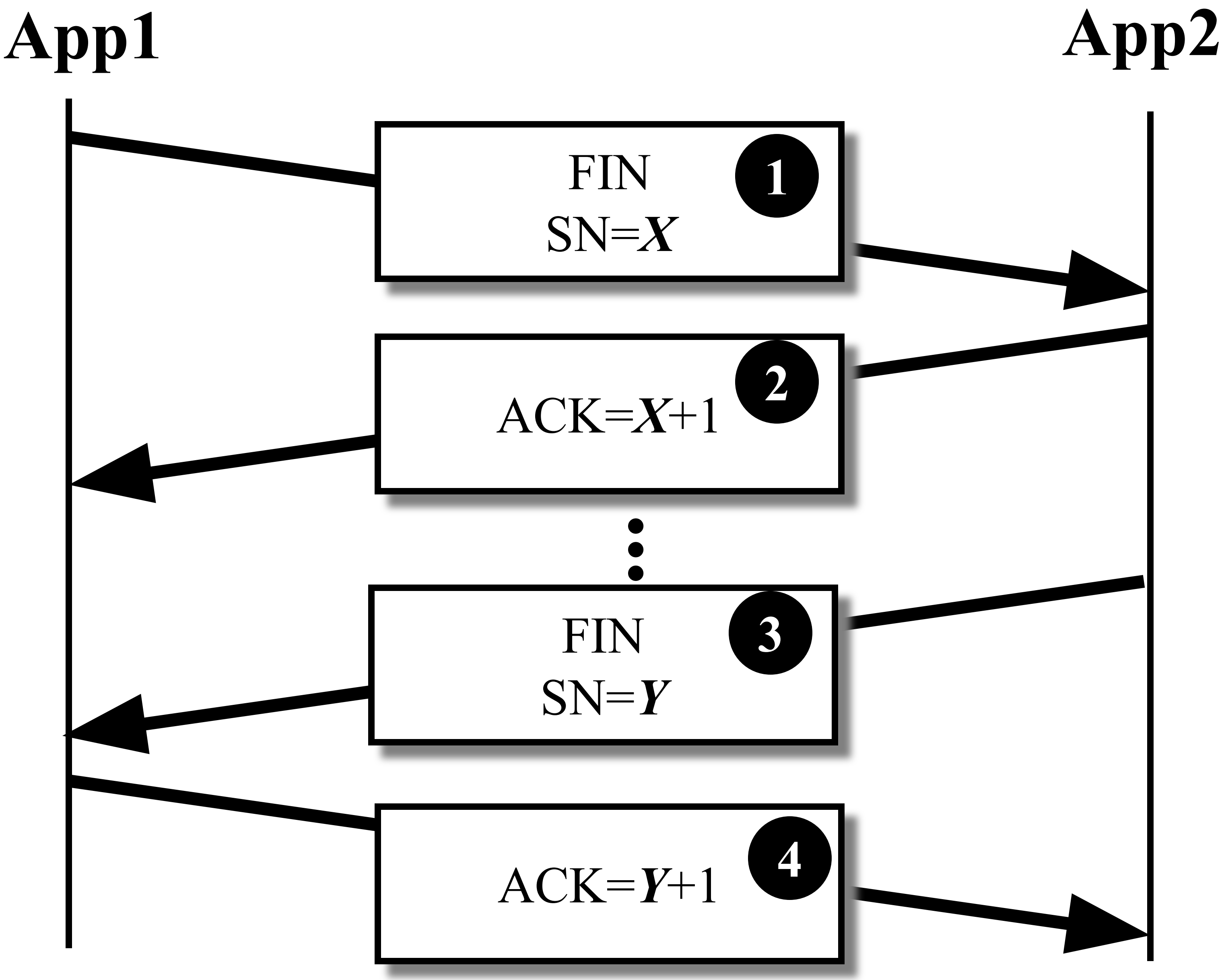
- Most TCP implementations will accept out-of-order segments (if there is room in the buffer).
- Once the missing segments arrive, a single ACK can be sent for the whole thing.
- Remember: IP delivers TCP segments, and IP in not reliable - IP datagrams can be lost or arrive out of order.

Termination

- The TCP layer can send a RST segment that terminates a connection if something is wrong.
- Usually the application tells TCP to terminate the connection politely with a FIN segment.

FIN

- Either end of the connection can initiate termination.
- A FIN is sent, which means the application is done sending data.
- The FIN is ACK'd.
- The other end must now send a FIN.
- That FIN must be ACK'd.


App1
App2
FIN
SN=X
1
2
ACK=X+1
FIN
SN=Y
3
4
ACK=Y+1

TCP Termination

1. App1: “I have no more data for you”.

2. App2: “OK, I understand you are done sending.”

   dramatic pause…

3. App2: “OK - Now I’m also done sending data”.

4. App1: “Roger, Over and Out, Goodbye, Astalavista Baby, Adios, It’s been real ...”

   camera fades to black ...

TCP TIME_WAIT

- Once a TCP connection has been terminated (the last ACK sent) there is some unfinished business:
  - What if the ACK is lost? The last FIN will be resent and it must be ACK'd.
  - What if there are lost or duplicated segments that finally reach the destination after a long delay?
- TCP hangs out for a while to handle these situations.

Test Questions

- Why is a 3-way handshake necessary?
  - HINTS: TCP is a reliable service, IP delivers each TCP segment, IP is not reliable.
- Who sends the first FIN - the server or the client?
- Once the connection is established, what is the difference between the operation of the server's TCP layer and the client's TCP layer?
- What happens if a *bad guy* can guess ISNs?